2. Klausur zur Vorlesung
Theoretische Grundlagen der Informatik
Wintersemester 2022/2023

Hier Aufkleber mit Matrikelnummer anbringen

- Bringen Sie den Aufkleber mit Ihrer Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrer Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Am Ende der Klausur sind zusätzliche Leerseiten. Fordern Sie zusätzliches Papier bitte nur an, falls Sie den gesamten Platz aufgebraucht haben.
- Die Tackernadel darf nicht gelöst werden.
- Als Hilfsmittel ist ein beschriebenes A4-Papier erlaubt.
- Einlesezeit: 15 min
  Bearbeitungszeit: 2 h

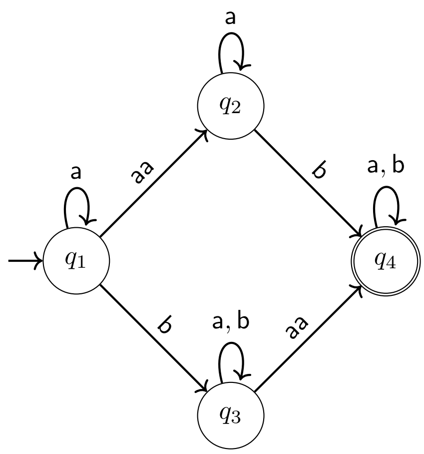
	Mögliche Punkte						Erreichte Punkte					
	a	b	c	d	e	Σ	a	b	c	d	e	Σ
Aufg. 1	2	2	3	3	–	10					–	
Aufg. 2	3	3	2	–	–	8				–	–	
Aufg. 3	3	1	2	5	3	14						
Aufg. 4	1	1	6	–	–	8				–	–	
Aufg. 5	1	2	1	4	3	11						
Aufg. 6	1	5	3	–	–	9				–	–	
Σ						60						

Problem 1: Block-NEA $2 + 2 + 3 + 3 = 10$ Punkte

Wir definieren einen k-*Block-NEA* wie einen NEA, mit der Änderung, dass pro Schritt bis zu k Zeichen gelesen werden dürfen. Ein k-Block-NEA hat also auch eine Zustandsmenge Q, einen Startzustand s, Endzustände $F \subseteq Q$ und ein Alphabet Σ. Die Übergangsfunktion δ kann bei jedem Übergang bis zu k Zeichen lesen.

Formal hat ein Übergang also die Form $\delta(q, w) = p$, wobei p und q Zustände sind und $w \in \Sigma^*$ ein Wort der Länge $|w| \leqslant k$ ist. Ein Wort w wird genau dann akzeptiert, wenn es eine Folge von Übergängen durch Zustände $q_0, q_1, \ldots, q_\ell$ mit $\delta(q_i, w_{i+1}) = q_{i+1}$ gibt, wobei $q_0 = s$ der Startzustand ist, $q_\ell \in F$ akzeptierend und $w = w_1 \cdot w_2 \cdot \ldots \cdot w_\ell$ die Konkatenation der jeweils gelesenen Wörter ist.

(a) Welche Sprache erkennt der folgende 2-Block-NEA?



(b) Geben Sie einen 3-Block-NEA an, der genau die Wörter über dem Alphabet $\{\mathsf{a}, \mathsf{b}\}$ erkennt, in denen das Teilwort aab eine gerade Anzahl oft vorkommt.

(c) Zeigen Sie, dass k-Block-NEAs gleich mächtig wie NEAs sind.

Betrachten wir nun nur Sprachen, deren Wörter ein Vielfaches von k lang sind. Wir interessieren uns für k-Block-NEAs, bei denen in jedem Schritt genau k Zeichen gelesen werden, diese Automaten nennen wir *vollständig*.

(d) Sei $\mathcal{A}$ ein k-Block-NEA. Beschreiben Sie, wie ein vollständiger k-Block-NEA $\mathcal{A}'$, der die gleiche Sprache wie $\mathcal{A}$ erkennt, konstruiert werden kann.

Hinweis: Es ist hilfreich, $\mathcal{A}$ zuerst in einen NEA umzuwandeln. Hierzu dürfen Sie (c) verwenden, auch wenn Sie die Teilaufgabe nicht bearbeitet haben. Konstruieren Sie dann daraus einen vollständigen k-Block-NEA.

Problem 2: Turingmaschinen $3 + 3 + 2 = 8$ Punkte

Wir definieren die Sprache

$$L_{\text{xor}} = \{x\#y \mid x, y \in \{\mathtt{0}, \mathtt{1}\}^*, |x| = |y| \text{ und } x \oplus y = \mathtt{1}^{|x|}\},$$

wobei $\oplus$ bitweises XOR darstellt. Für zwei Bits a und b gilt: $a \oplus b = 1 \iff a \neq b$.

Beispielsweise sind die Wörter 110#001 und 1#0 in L_{xor}, aber 11#0 und 011#010 sind nicht in L_{xor}.

Hinweis: Der Begriff Turingmaschine *meint eine Turingmaschine wie in der Vorlesung definiert, insbesondere mit nur einem Band und nur einem Kopf.*

(a) Beschreiben Sie eine Turingmaschine, die L_{xor} im $\mathcal{O}$-Kalkül möglichst effizient entscheidet. Geben Sie außerdem die Laufzeit im $\mathcal{O}$-Kalkül an.

Unser Ziel wird nun sein zu zeigen, dass es keine Turingmaschine gibt, die L_{xor} schneller als in quadratischer Zeit entscheidet. Dazu definieren wir die Sprache

$$L_{\text{dup}} = \{w\#w \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\} \ .$$

Für diese Sprache ist Folgendes bekannt: Es gibt keine Turingmaschine, die L_{dup} schneller als in quadratischer Zeit entscheidet. Formal bedeutet das, dass jede Turingmaschine, die L_{dup} entscheidet, im Worst Case $\Omega(n^2)$ Schritte benötigt, wobei n die Eingabelänge ist.

(b) Geben Sie eine Transformation f von L_{dup} nach L_{xor} an, die von einer Turingmaschine in Linearzeit berechenbar ist. Für alle Wörter w soll also

$$w \in L_{\text{dup}} \Longleftrightarrow f(w) \in L_{\text{xor}}$$

gelten. Beweisen Sie die Korrektheit und Laufzeit Ihrer Transformation f.

(c) Zeigen Sie, dass es keine Turingmaschine gibt, die L_{xor} schneller als in quadratischer Zeit entscheidet.

Problem 3: Grammatiken $3 + 1 + 2 + 5 + 3 = 14$ Punkte

Für eine Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_+$ bezeichnen wir eine Grammatik $G = (\Sigma, V, S, R)$ als $f(n)$-*Grammatik*, wenn es für jedes Wort $w \in L(G)$ eine Ableitung $S \xrightarrow{*} w$ in höchstens $f(|w|)$ Ableitungsschritten gibt. Außerdem bezeichnen wir eine Grammatik als $\mathcal{O}(f(n))$-*Grammatik*, wenn sie eine $g(n)$-Grammatik ist mit $g(n) \in \mathcal{O}(f(n))$.

(a) Zeigen Sie, dass die folgende Grammatik G mit $L(G) = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_0\}$ eine $\mathcal{O}(n^2)$-Grammatik, aber keine $\mathcal{O}(n)$-Grammatik ist.

$G = (\Sigma, V, S, R)$ mit $\Sigma = \{\mathsf{a}, \mathsf{b}\}, V = \{S, X, A, B\}$ und

$$
\begin{aligned}
R = \{ \quad S &\to \mathsf{a}BX \mid \varepsilon, \\
X &\to ABX \mid \varepsilon, \\
BA &\to AB, \\
\mathsf{a}A &\to \mathsf{aa}, \\
\mathsf{a}B &\to \mathsf{ab}, \\
\mathsf{b}B &\to \mathsf{bb} \\
\}
\end{aligned}
$$

(b) Geben Sie eine $\mathcal{O}(n)$-Grammatik für $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_0\}$ an und zeigen Sie, dass Ihre Grammatik tatsächlich eine $\mathcal{O}(n)$-Grammatik ist.

(c) Zeigen Sie, dass es für jede kontextfreie Sprache L eine $\mathcal{O}(n)$-Grammatik gibt, die L erzeugt.

(d) Zeigen Sie, dass eine Sprache L genau dann in NP ist, wenn es eine $p(n)$-Grammatik gibt, die L erzeugt, wobei p ein Polynom ist. Sie dürfen das folgende Theorem ohne Beweis verwenden.

Theorem 1. *Für jede nichtdeterministische Turingmaschine $\mathcal{M}$ gibt es eine Grammatik G mit $L(G) = L(\mathcal{M})$. Dabei hat die Grammatik G für jeden Berechnungspfad der Länge ℓ von $\mathcal{M}$, der ein Wort w akzeptiert, eine Ableitung $S \xrightarrow{*} w$ der gleichen Länge ℓ.*

(mehr Platz auf der nächsten Seite)

(e) Sei f eine berechenbare Funktion. Zeigen Sie: Wenn G eine $f(n)$-Grammatik ist, dann gibt es eine nichtdeterministische Turingmaschine, bei der jeder Berechnungspfad terminiert und die $L(G)$ akzeptiert.

Hinweis: Sie dürfen in dieser Teilaufgabe mehrere Bänder verwenden, sofern dies sinnvoll ist.

Problem 4: NP-Vollständigkeit $1 + 1 + 6 = 8$ Punkte

Wir betrachten das folgende Problem, von dem wir zeigen wollen, dass es NP-vollständig ist.

COLUMNFLIPPING

Gegeben: Eine $(2 \times n)$-Matrix mit Einträgen in $\mathbb{N}_0$, wobei $n \geqslant 1$

Frage: Für jede Spalte darf nun entschieden werden, ob die beiden Einträge vertauscht werden. Gibt es eine Menge S von Spalten, sodass das Vertauschen der beiden Einträge für jede Spalte in S dazu führt, dass die Summe beider Zeilen gleich ist?

Beispiel:

Die Matrix M ist eine Ja-Instanz, da die Wahl von S als erste und dritte Spalte dazu führt, dass die Summe beider Zeilen jeweils 11 ist.

$$M = \begin{pmatrix} 2 & 1 & 0 & 0 & 3 & 4 \\ 0 & 0 & 3 & 9 & 0 & 0 \end{pmatrix} \quad \rightsquigarrow \quad \begin{pmatrix} 0 & 1 & 3 & 0 & 3 & 4 \\ 2 & 0 & 0 & 9 & 0 & 0 \end{pmatrix} \quad \begin{matrix} \Sigma = 11 \\ \Sigma = 11 \end{matrix}$$

(a) Konstruieren Sie eine Nein-Instanz von COLUMNFLIPPING.

Sie wissen außerdem aus der Vorlesung, dass PARTITION NP-vollständig ist.

PARTITION

Gegeben: Eine endliche Menge M,
eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$

Frage: Existieren disjunkte Teilmengen $M_1, M_2 \subseteq M$ mit $M_1 \cup M_2 = M$ und $w(M_1) = w(M_2)$?

Für eine Menge X ist hierbei $w(X)$ definiert als $w(X) = \sum_{x \in X} w(x)$.

Wir zeigen nun schrittweise, dass COLUMNFLIPPING ebenfalls NP-vollständig ist.

(b) Zeigen Sie, dass COLUMNFLIPPING in NP liegt. Geben Sie dabei explizit an, was der Lösungsvorschlag des Orakels enthält.

(c) Geben Sie eine polynomielle Transformation von PARTITION zu COLUMNFLIPPING an und zeigen Sie, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist.

Hinweis: Betrachten Sie das Beispiel noch einmal.

- Geben Sie hier Ihre polynomielle Transformation an (ohne Beweis[1]):

- Zeigen Sie nun, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist:

[1] Falls Sie aus Versehen an dieser Stelle einen Beweis geschrieben haben, kennzeichnen Sie eindeutig, welcher Teil zur Transformation gehört und welcher Teil zum Beweis.

Problem 5: Approximation $\quad 1 + 2 + 1 + 4 + 3 = 11$ Punkte

Bei dem Optimierungsproblem Unit Square Cover geht es darum, Punkte in der Ebene durch möglichst wenige Einheitsquadrate (mit Seitenlänge 1) zu überdecken.

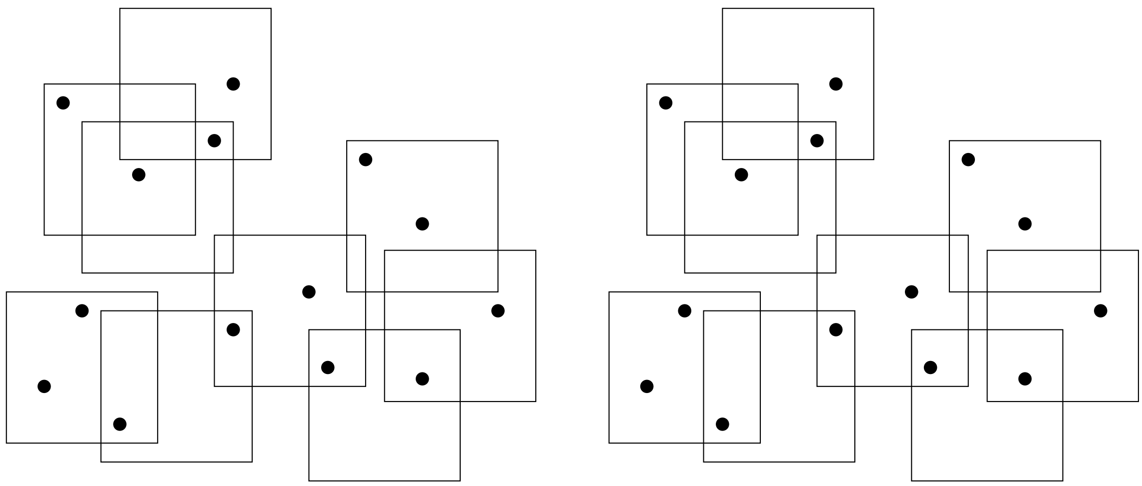
Unit Square Cover

Gegeben:
- Menge von Punkten $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ in der Ebene, wobei $x_i, y_i \geqslant 0$ für alle $1 \leqslant i \leqslant n$
- Menge von achsenparallelen (Seiten sind parallel zur x- oder y-Achse) Einheitsquadraten $Q = \{q_1, \ldots, q_k\}$ mit Seitenlänge 1

Gesucht: Kardinalitätsminimale Teilmenge $Q^* \subseteq Q$, sodass jeder Punkt in P von (mindestens) einem Quadrat aus Q^* überdeckt wird
Hinweis: Liegt ein Punkt auf dem Rand oder im Inneren eines gewählten Quadrats, wird dieser überdeckt.

(a) Markieren Sie für die Instanz in der folgenden Abbildung eine kardinalitätsminimale Teilmenge der gegebenen Einheitsquadrate, die die Punkte überdecken.

Falls Sie beide Kopien der Instanz bearbeiten, markieren Sie eindeutig die zu wertende Kopie.



Instanz für Teilaufgabe (a) (und eine zusätzliche Kopie)

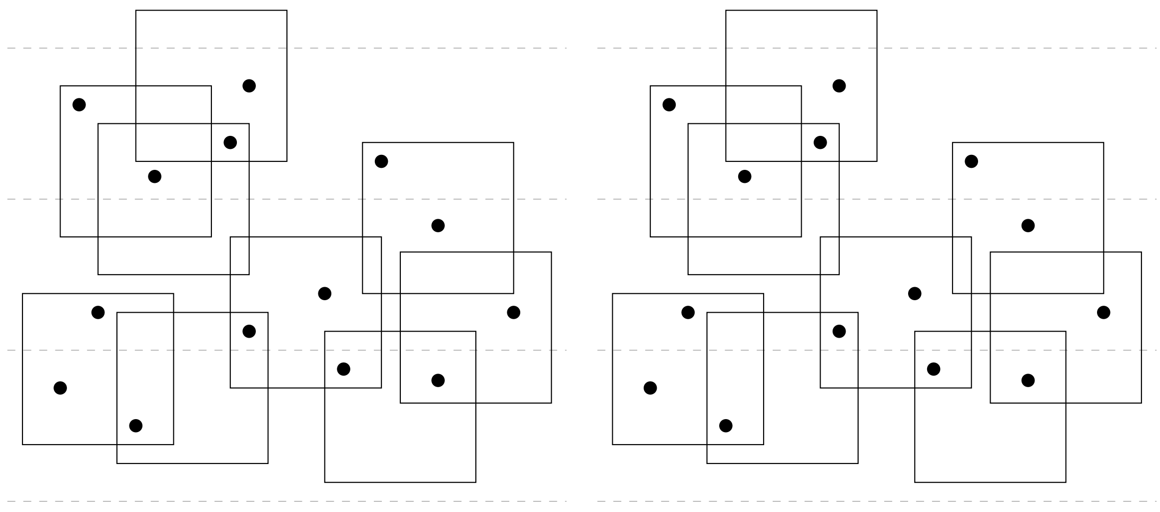
Betrachten Sie nun den polynomiellen Algorithmus $\mathcal{A}$.

Algorithmus $\mathcal{A}$: Unit Square Approx

Partitioniere das Koordinatensystem in Streifen $S_0, S_1, \ldots$ der Höhe 1 mit
$S_i = \{(x, y) \mid x \geqslant 0,\ y \in [i, i+1)\}$
Für *jeden Streifen S_i, der einen Punkt enthält*
  Bestimme eine optimale Lösung Q_i, die alle Punkte in S_i überdeckt
return $Q_0 \cup Q_1 \cup Q_2 \cup \ldots$ *(Vereinigung aller Q_i)*

(b) Markieren Sie für die Instanz in der folgenden Abbildung die Einheitsquadrate, die von $\mathcal{A}$ ausgewählt werden. Die Streifen sind schon vorgegeben und haben Höhe 1.

Falls Sie beide Kopien der Instanz bearbeiten, markieren Sie eindeutig die zu wertende Kopie.



Instanz für Teilaufgabe (b) (und eine zusätzliche Kopie)

Sei für eine Instanz I von Unit Square Cover $\mathrm{OPT}(I)$ eine optimale Lösung von I. Für jeden Streifen S_i definieren wir die Menge $\mathrm{OPT}_i \subseteq \mathrm{OPT}(I)$ von Quadraten, die in der optimalen Lösung von I einen Punkt in S_i überdeckt.

(c) Zeigen Sie, dass $|Q_i| \leqslant |\,\mathrm{OPT}_i\,|$ gilt. Dabei bezeichnet Q_i die Menge der Quadrate, die der Algorithmus $\mathcal{A}$ für Streifen S_i wählt.

(d) Zeigen Sie, dass $\mathcal{A}$ ein Approximationsalgorithmus für Unit Square Cover mit relativer Güte 2 ist. Sie müssen nicht zeigen, dass $\mathcal{A}$ in polynomieller Zeit läuft.

(mehr Platz auf der nächsten Seite)

(e) Sie dürfen annehmen, dass Unit Square Cover NP-vollständig ist. Zeigen Sie, dass es keinen polynomiellen Approximationsalgorithmus mit absoluter Güte $c \in \mathbb{N}$ gibt, falls $\mathsf{P} \neq \mathsf{NP}$.

Problem 6: Entscheidbarkeit $1 + 5 + 3 = 9$ Punkte

Sei $\Sigma = \{0, 1\}$ ein Alphabet. Sei $k \in \mathbb{N}_0$ beliebig, aber fest. Betrachten Sie die folgende Sprache:

$$L_k = \{\langle \mathcal{M} \rangle \colon |L(\mathcal{M})| \geqslant k\}$$

(a) Welchen maximalen Chomsky-Typ hat die Sprache für $k = 0$? Begründen Sie kurz.

Aus der Vorlesung wissen Sie, dass die universelle Sprache

$$L_u = \{\langle \mathcal{M} \rangle \# w \colon w \in L(\mathcal{M})\}$$

nicht entscheidbar ist.

(b) Zeigen Sie, dass L_k für $k > 0$ nicht entscheidbar ist, indem Sie von der universellen Sprache reduzieren.

(c) Zeigen Sie, dass L_k semi-entscheidbar ist, indem Sie eine deterministische Turingmaschine konstruieren, die L_k semi-entscheidet. Geben Sie dabei die Reihenfolge der Berechnungsschritte genau an. Sie müssen nicht beschreiben, wie Sie die Bänder verwalten.